Meet-in-the-Middle Attack on 4+4 Rounds of SCARF under Single-Tweak Setting

Siwei Chen$^{1,6}$, Kai Hu$^{2,7}$, Guozhen Liu$^3$, Zhongfeng Niu$^3$, Quan Quan Tan$^4$, Shichang Wang$^5$

1 School of Cyber Science and Technology, Hubei University, Wuhan, Hubei, China. chensiwei@hubu.edu.cn
2 School of Cyber Science and Technology, Shandong University, Qingdao, Shandong, China. kai.hu@sdu.edu.cn
3 School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China. niuzhongfeng@ucas.ac.cn, liuguozhen@ucas.ac.cn
4 Nanyang Technological University, Singapore. quanquan001@e.ntu.edu.sg
5 Key Laboratory of Cyberspace Security Defense, Institute of Information Engineering, CAS, Beijing, China. wangshichang01ie.ac.cn
6 Key Laboratory of Intelligent Sensing System and Security, Ministry of Education, Hubei University, Wuhan, Hubei, China.
7 Key Laboratory of Cryptologic Technology and Information Security, Ministry of Education, Shandong University, Jinan, China.

Abstract. SCARF, an ultra low-latency tweakable block cipher, is the first cipher designed for cache randomization. The block cipher design is significantly different from the other common tweakable block ciphers; with a block size of only 10 bits, and yet the input key size is a whopping 240 bits. Notably, the majority of the round key in its round function is absorbed into the data path through AND operations, rather than the typical XOR operations. In this paper, we present a key-recovery attack on a round-reduced version of SCARF with 4 + 4 rounds under the single-tweak setting. Our attack is essentially a Meet-in-the-Middle (MitM) attack, where the matching phase is represented by a system of linear equations. Unlike the cryptanalysis conducted by the designers, our attack is effective under both security requirements they have outlined. The data complexity of our attack is $2^{10}$ plaintexts, with a time complexity of approximately $2^{60.63}$ 4-round of SCARF encryptions. It is important to note that our attack does not threaten the overall security of SCARF.

Keywords: Low-Latency, Tweakable, SCARF, Meet-in-the-Middle, Single-Tweak

1 Introduction

Cache side-channel attacks such as the contention-based cache attack where PRIME+PROBE [10,17] is a typical example, has increasingly becoming a critical threat to the security of architectural level of desktop and server grade CPUs.
One popular countermeasure which increases the complexity of such attacks is the use of randomized cache architectures [18,11,12,15,19,13,14,16]. Recently, the first dedicated cache randomization cipher SCARF (Secure CAche Randomization Function) was proposed by cryptographers in [3]. As block size of this cipher is merely 10 bits, analyzing its security and cryptographic properties is both academically interesting and important to the real-world security of CPUs.

Due to the large performance gap between the CPU and the memory, modern CPUs store frequently accessed data in small memory modules called caches. For economic reasons, most modern processors divide the cache memory into multiple levels ranked from the smallest, fastest, but also the most expensive, L1 cache to the largest, slowest but also the cheapest L3 cache. When the CPU attempts to read to and from or write to a memory address, the caches are queried using this address first. If the data associated with the requested address is stored in the cache, we said that the data is cached, and a cache hit occurs. The data is then returned directly from the cache. In this case, the CPU does not need to wait for the reply from the main memory which is usually slower. If the data is not cached, a cache miss occurs and the data is loaded from memory.

To determine if a data is cached, part of the data address is stored as a tag along with the data. For the small L1, a fully-associative implementation is used where one can directly search the entire cache for the given tag. As for larger L2 and L3 caches that typically holds multiple megabytes of storage, searching the entire cache is usually infeasible and therefore, a set-associative strategy is popular. A set-associative cache can be visualized by a \( n \times m \) matrix. The cache is divided into \( n \) sets and each set contains \( m \) cache lines. A memory block is first mapped onto a set and then placed into any cache line of the set. When performing the mapping, a memory address is split into 3 different parts: a tag, an index and an offset. The tag and index are often used to compute a pseudorandom set index while the offset is used to indicate the specific position of the data inside the cache line.

Typically, L1 and L2 caches are private to each individual core, while L3 cache is shared among all the cores and therefore, it is possible for an attacker to perform a cross-core cache attack. One of these attacks is called contention-based attack where the PRIME+PROBE [10,17] attack is the most common example. In this attack, an attacker constructs a set of addresses in which all the addresses have the same index called the eviction set. Then, the attacker primes the cache set by accessing all addresses in the eviction set. Next, he triggers the victim’s program. Finally, the attacker re-accesses all addresses in the eviction set. Now, since reading the data from the main memory is slower than cache, the attacker can guess reliably if a cache miss has occurred. If it does, it indicates that the victim’s program has accessed the target address. The PRIME+PROBE technique has been used for key-recovery attacks against GnuPG [8].

A countermeasure against the contention-based cache attack is to use cache randomization. This increases the difficulty of the contention-based cache attacks [11,12,15,19,13,14,16]. Given an address, the randomization function is used to generate a pseudorandom index for the cache set. Recently, Canale et al.
proposed the first dedicated cache randomization cipher for this purpose called SCARF [3]. SCARF is an extremely low-latency tweakable block cipher. Note that the low-latency property is extremely crucial in this scenario as the cipher is in the critical path of the cache access. The tweakable feature comes from the fact that it takes the tag and the index as input, which is a popular way to perform cache randomization. SCARF accepts as input a 240-bit master key, a 48-bit (the length of the tag) tweak and 10-bit (the length of the index) plaintext and outputs a 10-bit pseudorandom index after 8 iterated round functions. To ensure that an eviction set is hard to construct, the pseudorandom index should not be able to be predicted trivially. However, under this setting, the ciphertext of SCARF, i.e., the output of the cache randomization cipher, is actually not visible to the attacker. As a result, the most important cryptographic property of SCARF is to resist the collision attack: the attacker should not have an advantage to find a pair of addresses such that they do not have the same index after applying SCARF compared to an ideal tweakable permutation. This leads to the Security Requirement 1 in Section 2.2. To encourage and facilitate the cryptanalysis, the designers also provided a stronger model which is stated in Security Requirement 2 which claimed that the attacker does not have an advantage to distinguish the composition of a decryption and an encryption function under different tweaks.

Since the design of SCARF is somehow aggressive for the extremely low-latency target, analyzing its security has become crucial for the future of this design paradigm. The designers have performed extensive cryptanalysis on SCARF, most of which are under the Security Requirement 2 and majority of the attention are catered to distinguishing attacks. Although their results showed that SCARF is strong enough against common attacks such as differential [2], linear [9], impossible differential [1], integral [6,4] and meet-in-the-middle [5] attacks, there is no non-trivial key-recovery attacks on SCARF even for round-reduced versions, to the best of our knowledge.

Our contributions. In this paper, we present a key-recovery attack on 4 + 4 rounds of SCARF. The primary attack is under Security Requirement 2, but one can easily transform it into an attack under Security Requirement 1 with some cheap pre-computations. Our attack is basically a Meet-in-the-Middle (MitM) attack. For $E_{T_2,T_1}(\cdot) = E_{T_2}^{-1} \circ E_{T_1}(\cdot)$, where $E$ is the 4 rounds of SCARF (as shown in Security Requirement 2), we first guess the 60-bit key for the first two rounds of $E_{T_1}$ and the last two rounds of $E_{T_2}^{-1}$ (note that they use the same 60-bit key), then we can peel off the outer rounds and established a system of linear equations for the inner 2 + 2 rounds thanks to the unique $G$ function used by SCARF. This system of linear equations serves as a filter for the guessed 60-bit key. The filtering effect is evaluated by a theoretical way, which is enough for filtering out all wrong keys. Eventually, we can recover all the 120-bit keys with approximately $2^{60.63}$ 4-round SCARF queries and $2^{10}$ plaintexts (i.e., the whole codebook) under Security Requirement 2. Our attack can also work under the security requirement 1 with cheap pre-computations that costs $2^{21}$ SCARF encryptions.
Outline. The following paper is organized as follows. In Section 2, we introduce the specification of SCARF and its two security requirements. Section 3 introduces our main attack and does the complexity analysis. Section 4 concludes this paper.

2 Preliminaries

2.1 Description of SCARF

SCARF (Secure Cache Randomization Function) is a low-latency tweakable block cipher designed for cache randomization. Due to its unique functionality requirements, the parameters of the primitive, as well as the security models, differ from those typically encountered by cryptanalysts. Therefore, we will introduce the various specifications of SCARF alongside the properties and policies for cache management. Firstly, under the write-back policy, which requires newly modified cache entries to be written back to the same address in the main memory, SCARF has to be invertible (an alternative would be to store the original index as part of the tag; however, this approach incurs extra memory overhead). Secondly, working under the assumption that the usage is a typical modern desktop-grade CPU, which usually has a 64-bit architecture, the address for the cache is split into three parts, a 48-bit tag, a 10-bit index, and a 6-bit offset. The 48-bit tag is used to identify which block of the main memory is currently inside the cache, the 10-bit index is used to identify the block number in the cache, and the 6-bit offset is used to find the exact location of the word within the block. From the attacker’s perspective, he is only interested in finding collisions on the index bits; therefore, SCARF is designed to randomize the 10-bit index. The other parameters are: 48-bit tweak (tag), 10-bit (index) block size and 240-bit secret key. The total number of rounds for SCARF is 8. An overview of the encryption algorithm can be found in Figure 1. The round function $R_1$ and the last round function $R_2$ can be found in Figure 2 and Figure 3 respectively.

Round Functions $R_1$ and $R_2$. The round function $R_2$ is just the function $R_1$ with an additional operation where we do not swap $x_L$ and $x_R$ at the output. Thus, we will focus the description just on $R_1$. The round function $R_1$ takes in a 10-bit input $X = (x_L||x_R)$ and 30-bit subkey $k = (k_6||k_5||k_4||k_3||k_2||k_1)$ where each sub-component is 5-bit in length. Let $\tau_i(x)$ defines the left rotation of $x$ by $i$, i.e. $\tau_i(x) = x \ll i$, then the round function $R_1$ updates $(x_L, x_R)$ as follows:

$$y = G(x_L, k_1, k_2, k_3, k_4, k_5) \oplus x_R,$$

$$x_R = S(x_L \oplus k_6),$$

$$x_L = y$$

where $G$ is

$$G(x, k_1, k_2, k_3, k_4, k_5) := \bigoplus_{i=0}^{4} (\tau_i(x) \wedge k_{i+1}) \oplus (\tau_1(x) \wedge \tau_2(x))$$
and $S$ is the 5-bit substitution box defined by

$$S(x) := \left( (\tau_0(x) \lor \tau_1(x)) \land (\tau_3(x) \lor \tau_4(x)) \right) \oplus \left( (\tau_0(x) \lor \tau_2(x)) \land (\tau_2(x) \lor \tau_3(x)) \right)$$

**Tweakey Schedule.** The tweakey schedule takes in a 48-bit tweak $T$ and 240-bit secret key $K$ where $K = K^{(4)} || K^{(3)} || K^{(2)} || K^{(1)}$ and generates four 60-bit sub-tweakeys $T^{(1)}, T^{(2)}, T^{(3)}, T^{(4)}$. Each sub-tweakey $T^{(i)} = (r_{k^{(2i)}} || r_{k^{(2i-1)}})$ will be used as the round key for round $2i - 1$ and $2i$ respectively. The tweakey
The round function $R_1$.

Fig. 2. The round function $R_1$.  

Fig. 3. The round function $R_2$. 

The schedule is as follows:
\[
T^{(1)} = \text{expansion}(T) \oplus K^{(1)},
\]
\[
T^{(2)} = \sum(SL(T^{(1)})) \oplus K^{(2)},
\]
\[
T^{(3)} = SL(\pi(SL(T^{(2)})) \oplus K^{(3)})),
\]
\[
T^{(4)} = SL(\sum(T^{(3)}) \oplus K^{(4)}),
\]

where
\[
\text{expansion}(T) = 0 \parallel T[48] \parallel T[47] \parallel T[46] \parallel T[45] \parallel
\]
\[
0 \parallel T[44] \parallel T[43] \parallel T[42] \parallel T[41] \parallel
\]
\[
\vdots
\]
\[
\]

the function $SL$ is a substitution layer that applies $12$ $S$ in parallel,  
the function $\sum$ is a linear function defined as
\[
\sum(x) := x \oplus \tau_6(x) \tau_{12}(x) \oplus \tau_{19}(x) \oplus \tau_{29}(x) \oplus \tau_{43}(x) \oplus \tau_{51}(x),
\]
and the function $\pi$ performs a bit permutation, where it updates the bits from position $i$ to position $p(i)$. The map $p$ is given in Table 1.

2.2 Security Models

In a real-world setting, the attacker is assumed to be able to determine whether each query access results in a cache hit or a cache miss based on the timing. In


<table>
<thead>
<tr>
<th>$i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p(i)$</td>
<td>1</td>
<td>6</td>
<td>11</td>
<td>16</td>
<td>21</td>
<td>26</td>
<td>31</td>
<td>36</td>
<td>41</td>
<td>46</td>
<td>51</td>
<td>56</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$i$</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
<th>22</th>
<th>23</th>
<th>24</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p(i)$</td>
<td>2</td>
<td>7</td>
<td>12</td>
<td>17</td>
<td>22</td>
<td>27</td>
<td>32</td>
<td>37</td>
<td>42</td>
<td>47</td>
<td>52</td>
<td>57</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$i$</th>
<th>25</th>
<th>26</th>
<th>27</th>
<th>28</th>
<th>29</th>
<th>30</th>
<th>31</th>
<th>32</th>
<th>33</th>
<th>34</th>
<th>35</th>
<th>36</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p(i)$</td>
<td>3</td>
<td>8</td>
<td>13</td>
<td>18</td>
<td>23</td>
<td>28</td>
<td>33</td>
<td>38</td>
<td>43</td>
<td>48</td>
<td>53</td>
<td>58</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$i$</th>
<th>37</th>
<th>38</th>
<th>39</th>
<th>40</th>
<th>41</th>
<th>42</th>
<th>43</th>
<th>44</th>
<th>45</th>
<th>46</th>
<th>47</th>
<th>48</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p(i)$</td>
<td>4</td>
<td>9</td>
<td>14</td>
<td>19</td>
<td>24</td>
<td>29</td>
<td>34</td>
<td>39</td>
<td>44</td>
<td>49</td>
<td>54</td>
<td>59</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$i$</th>
<th>49</th>
<th>50</th>
<th>51</th>
<th>52</th>
<th>53</th>
<th>54</th>
<th>55</th>
<th>56</th>
<th>57</th>
<th>58</th>
<th>59</th>
<th>60</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p(i)$</td>
<td>5</td>
<td>10</td>
<td>15</td>
<td>20</td>
<td>25</td>
<td>30</td>
<td>35</td>
<td>40</td>
<td>45</td>
<td>50</td>
<td>55</td>
<td>60</td>
</tr>
</tbody>
</table>

Cryptographic terms, this is akin to the attacker being able to detect if two chosen plaintexts collide. Note that the attacker is unable to observe the ciphertexts, prevent them from using this information for any adaptive changes during the attack. This scenario is formally captured in Security Requirement 1 (see Figure 4).

**Security Requirement 1 ([3])** Let $O_{\text{real}}$ be the oracle in the real world that takes addresses $(x_1, T_1)$ and $(x_2, T_2)$, and returns 1 if $E_{T_1}(x_1) = E_{T_2}(x_2)$ and 0 otherwise, where $E$ is SCARF. Let $O_{\text{ideal}}$ be the oracle in the ideal world that takes addresses $(x_1, T_1)$ and $(x_2, T_2)$, and returns 1 if $\Pi_{T_1}(x_1) = \Pi_{T_2}(x_2)$ and 0 otherwise, where $\Pi$ is a tweakable random permutation with the same input/tweak/output lengths to SCARF. An adversary is allowed to make at most $2^{40}$ queries. Then, the adversary running in time at most $2^{80}$ cannot distinguish the real from the ideal world.

While Security Requirement 1 is sufficient to cover all the security requirements, current cryptanalysis does not have the necessary tools to analyze it yet. Hence, the designers proposed Security Requirement 2, which fits a more conventional scenario for cryptanalysts.

**Security Requirement 2 ([3])** Let $O_{\text{real}}$ be the oracle in the real world that takes a plaintext $P$ and a pair of tweaks $T_1, T_2$ as inputs and returns $C$ such that $C = E_{T_2}^{-1} \circ E_{T_1}(P)$, where $E$ is SCARF. Let $O_{\text{ideal}}$ be the oracle in the ideal world that takes a plaintext $P$ and a tweakable random permutation with the same input/tweak/output lengths to SCARF. An adversary is allowed to make at most $2^{40}$ queries. Then, the adversary running in time at most $2^{80}$ cannot distinguish the real from the ideal world.

The idea of the new scenario brought by Security Requirement 2 is that when we have a collision, i.e. $E_{T_1}(P_1) = C_1 = C_2 = E_{T_2}(P_2)$ and $E$ being invertible, we must have $E_{T_2}^{-1} \circ E_{T_1}(P_1) = P_2$. We can then define $\tilde{E}_{T_1, T_2} := E_{T_2}^{-1} \circ E_{T_1}$, as if it is another block cipher, albeit, with twice the number of rounds (see Figure 5). Note
that both security requirements have the same query complexities. However, in order to construct $\tilde{O}$ from $O$, multiple queries to $O$ are required for a single query to $\tilde{O}$ (we need to query until there is a collision). Thus, Security Requirement 2 is said to be much stronger than Security Requirement 1.

3 Attack Process

In this paper, we describe our MitM attacks on the 4+4 rounds of SCARF. Our attack is a single-tweak attack such that it works for any tweak. As we mentioned, our attack works under the Security Requirement 2 but it can be transformed into an attack under Security Requirement 1 as well. We will first focus on the Security Requirement 2 then, introduce the transformation that allows us to target Security Requirement 1.

3.1 Overview of Our Attack Strategy

Under the Security Requirement 2, we work with $\tilde{E}_{K,T,T'} = E_{K,T'}^{-1} \circ E_{K,T}$ where $E_{K,T}$ is an instance of (round reduced) SCARF encryption with a tweak $T$ and
secret key $K$ while $E_{K,T'}^{-1}$ is an instance of (round reduced) SCARF decryption with a tweak $T'$ and secret key $K$ (note that $E_{K,T}$ and $E_{K,T'}^{-1}$ use the same $K$). Given a plaintext $P_1$, although we do not know the ciphertext of SCARF encryption, i.e., $E_{K,T}(P_1)$ under $(K,T)$, Security Requirement 2 allows us to gain knowledge of another plaintext $P_2 = E_{K,T,T'}(P_1)$. Thus, we can treat $E_{K,T,T'}$ as a tweakable block cipher where $(T,T')$ is its tweak and $K$ is its master key. The task of this project is to attack $E_{K,T,T'}$ with 4 rounds each for $E_{K,T}$ and $E_{K,T}^{-1}$, which we abbreviated as $4 + 4$ rounds of SCARF. Specifically,

$$E_{K,T} = R_2 \circ R_1 \circ \cdots \circ R_1 \quad \text{and} \quad E_{K,T}^{-1} = R_2^{-1} \circ \cdots \circ R_1^{-1},$$

the target of our attack is then

$$\tilde{E}_{K,T,T'} = R_2^{-1} \circ R_1^{-1} \circ \cdots \circ R_1^{-1},$$

where $R_1$ and $R_2$ are the round functions as depicted in Figure 2 and 3. Recall that in Section 2, the round tweakeys used in $E_{K,T}$ and $E_{K,T}^{-1}$, are $(rk_1, r_2, r_3, r_4)$ and $(rk'_1, rk'_2, r_1, r_2)$ respectively. According to SCARF’s tweak schedule,

$$(rk_1, r_2) = \text{expansion}(T) \oplus K^{(1)}$$

$$(rk'_1, r'_2) = \text{expansion}(T') \oplus K^{(1)}.$$ 

As $T$ and $T'$ are known to us, if we guess the 60-bit $K^{(1)}$, then $rk_1, r_2$ and $rk'_1, r'_2$ are all known. We are able to peel off the outer two rounds of $E_{K,T}$ and $E_{K,T}^{-1}$. Then, by choosing the appropriate messages, a system of linear equations can be established for the inner 2 + 2 rounds of $E_{K,T,T'}$. We can then check if our guess of $K^{(1)}$ is correct by observing whether the system of linear equations is consistent. This serves as a filter to remove the wrong $K^{(1)}$ guesses.

Before we dive deeper into more details of our attack, we first discuss the probability that a system of randomly-chosen linear equations is consistent. This directly affect the effectiveness of the filter to remove the wrong key guesses.

### 3.2 On the Effectiveness of the Filter in the Match Phase

For the inner 2 + 2 rounds of SCARF, a linear system with $m$ equations will be constructed. We denote it by $A \cdot x = b$ where $A$ is an $m \times n$ binary matrix (we assume $m \geq n$, i.e., it is an overdetermined linear system), and $b$ is an $m$-dimensional binary vector. The vector $x$ is an $n$-dimensional binary vector that corresponds to the corresponding round tweakey. Whether the linear system is consistent serves as a filter for the guessed $K^{(1)}$. To evaluate the complexity, we need to figure out the effectiveness of the filter defined as follows:

**Definition 1.** Given a linear system $A \cdot x = b$, where $A \in \mathbb{F}_2^{m \times n}$ with $m \geq n$ is a random binary matrix and $b \in \mathbb{F}_2^n$ is a random binary vector. The **effectiveness** of the filter (i.e., satisfying the $A \cdot x = b$ condition), is defined as $e = m - p$, where $p = \text{rank}(A)$. 

When the guessed $K^{(1)}$ is correct, we can always find $x$ to satisfy all the equations in the system. However, for a wrongly guessed $K^{1}$, the linear system is satisfied by random. Suppose the rank of $A$ is $p \leq n$ (note we have assumed $m \geq n$), then we can transform the argument matrix $[A|b]$ into a triangle shape, denoted by $[A'|b']$, where the first $p$ rows of $A'$ is non-zero while the remaining $m - p$ rows of $A$ are zero. Obviously, only if the corresponding $m - p$ elements of $b'$ are zero, the linear system is consistent.

**Theorem 1.** For a system of linear system $A \cdot x = b$, $m \geq n$, where $A \in \mathbb{F}_2^{m \times n}$ is a random binary matrix and $b \in \mathbb{F}_2^n$ is a random binary vector, the expectation $e$ is then

$$E(e) \geq \sum_{p=1}^{n} \left( (m-p) \cdot \prod_{i=1}^{p} \left( 1 - 2^{i-1-m} \right) 2^{(p-m) \times (n-p)} \right).$$

Before we go into the proof, we briefly describe the sketch of the proof to understand it in a more intuitive way. The idea is to first reduce $A$ to its row-echelon form (see Equation (1)). Since we are working on a random binary matrix, we can assume that after performing the same row operations on $b$, it will still be a random vector. Assuming that the rank of the matrix $A$ is $p$, for the system of linear equations to be consistent, we need to ensure that $b_{p+1} = b_{p+2} = \ldots = b_m = 0$. With that, we will first find the probability that the rank of $A$ is $p$ in a lemma.

**Lemma 1.** Given a random binary matrix $A \in \mathbb{F}_2^{m \times n}$, $m \geq n$, where every element of $A$ follows a uniform distribution, we define the random variable $\text{rank}(A)$ as the rank of $A$. Then, for $1 \leq p \leq n$, we have

$$\Pr[\text{rank}(A) = p] \geq \prod_{i=1}^{p} \left( 1 - 2^{i-1-m} \right) 2^{(p-m) \times (n-p)}$$

**Proof.** As the rank of $A$ equals to the rank of its transpose $A^T$, we consider $\text{rank}(A^T)$ instead where now the number of rows is less than or equal to the number of columns. Let $V = (v_1, \ldots, v_n)$ be the ordered set of row vectors of $A^T$. We define $B_p$ as the event that the first $p$ vectors in $V$ are linearly independent where $1 < p \leq n$. Then, we have the following recursive relations between the
probabilities involving $B_p$ and $B_{p-1}$. 

\[
\Pr[B_p] = \Pr[B_p \cap B_{p-1}]
= \Pr[B_p | B_{p-1}] \cdot \Pr[B_{p-1}]
= (1 - 2^{p-1-m}) \cdot \Pr[B_{p-1}].
\]

Starting from $\Pr[B_1] = (1 - 2^{-m})$ (the vector should be non-zero), we can recursively apply the formula

\[
\Pr[B_p] = \prod_{i=1}^{p} (1 - 2^{i-1-m}).
\]

To find the probability that the rank of $A$ is exactly $p$, we use the fact that $\Pr[\text{rank}(A) = p]$ is at least the probability of both $B_p$ and the remaining $n - p$ vectors lying in the $p$-dimensional space spanned by the vectors in $B_p$, i.e.,

\[
\Pr[\text{rank}(A) = p] \geq \Pr[(\text{rank}(A) = p) \cap B_p]
= \Pr[(\text{rank}(A) = p) | B_p] \cdot \Pr[B_p]
= (2^{p-m})^{n-p} \cdot \Pr[B_p]
= \prod_{i=1}^{p} (1 - 2^{i-1-m}) \cdot 2^{(p-m) \times (n-p)}
\]

Now we can prove Theorem 1 easily.

**Proof (Proof of Theorem 1).** $E(e)$ is calculated by

\[
E(e) = \sum_{p=1}^{n} [(m - p) \Pr[\text{rank}(A) = p]],
\]

summing up the contributions to the effectiveness from each possible rank $p$ weighted by the probability that $\text{rank}(A) = p$. By Lemma 1, we have

\[
E(e) \geq \sum_{p=1}^{n} \left[(m - p) \cdot \left(\prod_{i=1}^{p} (1 - 2^{i-1-m}) \cdot 2^{(p-m) \times (n-p)}\right)\right].
\]

We can now relate the effectiveness of the filter with the probability that the system of linear equations is consistent. Let $C$ be the event that a random system of linear binary equations $A \cdot x = b$ is consistent. We can approximate

\[
\Pr[C] \approx 2^{-E(e)}.
\]

To capture more of the variances in the probability, we can even upper bound $\Pr[C]$ by $2^{-E(e)}$. From Theorem 1, we know that
\[
\log_2(\Pr[C]) \leq - \sum_{p=1}^{n} (m - p) \cdot \left( \prod_{i=1}^{p} (1 - 2^{i-1-m}) \cdot 2^{(p-m) \times (n-p)} \right).
\]

Remark 1. The exact relationship between \(E(e)\) and \(\Pr[C]\) can be complex due to the nature of how effectiveness propagates in the random system. The approximation \(\Pr[C] \approx 2^{-E(e)}\) provides an intuitive understanding of how the probability of consistency in the system of linear equation decreases exponentially with the expected effectiveness.

3.3 Key Recovery Attack on 4 + 4 Rounds of SCARF

In this section, we will illustrate our attack in detail. The linear system can be constructed thanks to the following important observation on the function \(G\).

Lemma 2. Suppose that the input and output of \(G\) are known and the input is non-zero, then, we can obtain 5-bit of key information.

Proof. Let \(x = (x_1, x_2, x_3, x_4, x_5)\) and \(y = (y_1, y_2, y_3, y_4, y_5)\) be the 5-bit input and output of \(G\) respectively. Let \((k_1, k_2, k_3, k_4, k_5)\) be the 25-bit key involved into the \(G\) function where \(k_i\) can be further represented by \(k_i = (k_{i,1}, k_{i,2}, k_{i,3}, k_{i,4}, k_{i,5})\) for \(1 \leq i \leq 5\). Then, we have the following 5 equations according to the specification of \(G\),

\[
\begin{align*}
y_1 &= x_1 k_{1,1} \oplus x_2 k_{2,1} \oplus x_3 k_{3,1} \oplus x_4 k_{4,1} \oplus x_5 k_{5,1} \oplus x_2 x_3 \\
y_2 &= x_2 k_{1,2} \oplus x_3 k_{2,2} \oplus x_4 k_{3,2} \oplus x_5 k_{4,2} \oplus x_1 k_{5,2} \oplus x_3 x_4 \\
y_3 &= x_3 k_{1,3} \oplus x_4 k_{2,3} \oplus x_5 k_{3,3} \oplus x_1 k_{4,3} \oplus x_2 k_{5,3} \oplus x_4 x_5 \\
y_4 &= x_4 k_{1,4} \oplus x_5 k_{2,4} \oplus x_1 k_{3,4} \oplus x_2 k_{4,4} \oplus x_3 k_{5,4} \oplus x_5 x_1 \\
y_5 &= x_5 k_{1,5} \oplus x_1 k_{2,5} \oplus x_2 k_{3,5} \oplus x_3 k_{4,5} \oplus x_4 k_{5,5} \oplus x_1 x_2
\end{align*}
\]

(2)

When \(x \neq 0\), each linear equation in Equation (2), involves five (different) independent key bits. Thus, one bit of key information can be obtained from each equation and five bits of information about the key can be obtained. \(\Box\)

Corollary 1. Suppose we have five input and output pairs of a \(G\) function that can generate 5 linearly independent linear equations, then we can obtain the whole information about the 25-bit key involved in \(G\).

The attack covers 4 rounds of the encryption function \(E_{K,T}\) and 4 rounds of the decryption function \(E_{K,T}^{-1}\), where \(T \neq T'\). The master keys for the first 4 rounds are denoted as \(K^{(1)} \in F_2^{60}\) and \(K^{(2)} \in F_2^{60}\). For the encryption \(E_{K,T}\), the input into the round \(R^{(r)}\) is denoted by \((x_L^{(r)}, x_R^{(r)}) \in F_2^{5 \times 2}\), where \(1 \leq r \leq 4\). For the decryption \(E_{K,T}^{-1}\), the input into the round \(R^{(r)}\) is denoted by \((x_L^{(r)}, x_R^{(r)}) \in F_2^{5 \times 2}\) where \(4 \geq r \geq 1\) (refer to Figure 6). The sub-tweakeys for encryption and decryption are denoted as \(k^{(r)}\) and \(k^{(r)}\), with \(1 \leq r \leq 4\). In each round, the sub-tweakey \(k^{(r)}\) (resp. \(k^{(r)}\)) are further denoted by \(k^{(r)}_{i} \in F_2^{60}\) (resp. \(k^{(r)}_{i} \in F_2^{60}\)) where \(1 \leq i \leq 6\) to indicate how each 5-bit sub-tweakey is incorporated to the round.
Preparation phase. Based on Lemma 2 and Corollary 1, if we know the input and output of $G$ (up to some guessed keys), we can construct the linear system of equations with the key bits involved as the variables. For the 4 + 4 rounds of SCARF, we first choose the $G$ of $R^{(3)}$ round as the target. When $x_L^{(3)} = 0$ and $x_R^{(3)} \neq 0$ are known, we can know the value of $x_L^{(4)}$ as well as $x_R^{(4)}$. Selecting the value of $(x_L^{(3)}, x_R^{(3)}) = (0, a)$ where $a \in \mathbb{F}_2^{10}$ involves creating a set of precomputed values that help in predicting the intermediate states via encryption for the input.

\footnote{Note that our attack is on the first 4 rounds of $E_{K,T}$, so $k^{(4)}_b = k^{(4)}_b$ according to the tweak key schedule. If we attack the last 4 rounds of $E_{K,T}$, we need to guess 6 bits more for $k^{(4)}_b \oplus k^{(4)}_b$ to obtain $x_L^{(4)}$.}
and decryption for the output. Concretely, we construct the full codebook of the plaintext pairs \((P_1, P_2) \in \mathbb{F}_2^{10} \times \mathbb{F}_2^{10}\) such that \(E_{K,T,T}(P_1) = P_2\), or more compactly, \(\tilde{E}_{K,T,T}(P_1) = P_2\). Note that, given that each pair of \((P_1, P_2)\) is uniquely determined, we will have the full codebook of size \(2^{10}\).

Assume that we have guessed a value for \(K^{(1)}\), for any \((x^{(3)}_L, x^{(3)}_R)\), we are able to decrypt it to obtain \((x^{(2)}_L, x^{(2)}_R)\) and \((x^{(1)}_L, x^{(1)}_R) = P_1\) using the sub-tweakeys derived from \(K^{(1)}\). Then, we can obtain the \(P_2\) corresponding to the \(P_1\) from the pre-computed table. Finally, we can encrypt \(P_2\) using the sub-tweakeys again to obtain \((x^{(3)}_L, x^{(3)}_R)\) (see Figure 6). Using this technique, as long as we start from \((x^{(3)}_L, x^{(3)}_R)\) and guess a \(K^{(1)}\), we can peel off the outer two rounds and focus on the inner 2 + 2 rounds.

**Key recovery phase.** Now we introduce the 4 + 4 rounds key recovery attack on \(\tilde{E}_{K,T,T}(P_1) = P_2\) in the single-tweak setting. Note that in Step 4 and Step 5, the filters used are labeled as Filter 1 and Filter 2 in Figure 6, respectively.

1. Let \((x^{(3)}_L, x^{(3)}_R) = (0, a)\) where \(a \in \mathbb{F}_2^3\). For each \(K^{(1)} \in \mathbb{F}_2^{10}\) and \((x^{(3)}_L, x^{(3)}_R)\) guess, we compute the corresponding \((x^{(3)}_L, x^{(3)}_R)\).

2. Encrypt and decrypt \((x^{(3)}_L, x^{(3)}_R)\) forward by 2 + 1 rounds, i.e., the third and fourth round of \(E_T\), denoted by \(R^{(3)}\) and \(R^{(4)}\), and the fourth round of \(E_T^{-1}\), denoted by \(R^{(4)}\). The left input of \(R^{(4)}\), i.e., \(x^{(4)}_L\), can be obtained as follows:
   (a) As \(x^{(3)}_L = 0\), we must have \(x^{(4)}_L = x^{(3)}_R\).
   (b) The left output of \(R^{(4)}\) is then \(S(x^{(3)}_R) \oplus k^{(4)}_6\) and the left output of the Sbox of \(R^{(4)}\) is \(S(x^{(3)}_R) \oplus \delta\) where \(\delta = k^{(4)}_6 \oplus k^{(4)}_6\). Given that \(T, T', K^{(1)}\) are known, we know that \(\delta = k^{(4)}_6 \oplus k^{(4)}_6 = \sum(SL(T \oplus K^{(1)})) \oplus \sum(SL(T' \oplus K^{(1)}))\).
   (c) At the output of \(R^{(3)}\), we get \(x^{(4)}_L = S^{-1}(S(x^{(3)}_R) \oplus \delta)\).
   (d) Lastly, since we have the value for \(x^{(3)}_R\), the output of \(G\) in \(R^{(3)}\) is \(x^{(3)}_R \oplus S^{-1}(S(x^{(3)}_R) \oplus \delta)\).

3. At this point, we have obtained all the inputs and outputs for \(G\) except for the 25 bits sub-tweakey. When \(x^{(3)}_L\) is non-zero, 5 out of the 25-bit sub-tweakey information (i.e., \(k^{(3)}_i \in \{1, \ldots, 5\}\)) can be obtained according to Lemma 2. Naturally, as stated in Corollary 1, if we add more of such equations, we can retrieve the full 25-bit sub-tweakey.

4. When we have the right key guess of \(K^{(1)}\), the system must be consistent and when we have the wrong key guess, the linear system will be inconsistent with a high probability. As discussed in Section 3.2, the probability of a random linear system being consistent is bounded by

\[
\Pr[C] \leq 2^{-E(c)} \leq 2^{-\sum_{p=1}^{n} \binom{m-p}{n-p} \cdot \left(\Pi_{i=1}^{n} (1 - 2^{p-1} - m) \cdot \alpha^{(p-n)} \right)}.
\]

\(\footnote{This scenario assumes that we are attacking the first 4 rounds of SCARF, where \(\delta\) is known. However, we also acknowledge that if the attack is to be applied to round 5 to 8 instead, then \(\delta\) will not be known due to the structure of the key schedule and would have to be guessed, adding an additional \(2^2\) time complexity to this step.}
For instance, when \( m = 16 \) (i.e., we construct 16 equations on the 5 bit sub-tweakey of \( k^{(3)}_i \) \( \forall i \in \{1, ..., 5\} \) by choosing 16 different \((x^{(3)}_L, x^{(3)}_R)\) states), and \( n = 5 \), we have

\[
\Pr[C] \leq 2^{-10.998}.
\]

Note that in total, we have 5 different system of linear equations, each represented by a constraint in Equation (2). Hence, the capability of the filter on \( R^{(3)} \) is bounded by

\[
\Pr[C] \leq 2^{-5 \cdot E(e)} \leq 2^{-5 \times 10.998} = 2^{-54.99}.
\]

This means that after this filtering step, only \( 2^{60-54.99} = 2^{5.01} \) \( K^{(1)} \) guesses will survive.

5. Now, we will shift our attention to find the sub-tweakeys related to \( R^{(4)} \). We will first guess \( k^{(5)}_6 \in \mathbb{F}_2^5 \). Together with the remaining keys from the previous step, we now have \( 2^{10.01} \) key candidates. Since \( K^{(1)} \) is known (guessed), we will be able to obtain a corresponding guess for \( k^{(3)}_6 \). First, we choose a pair of \((x^{(3)}_L, x^{(3)}_R)\) such that \( G(x^{(3)}_L) = x^{(3)}_R \).10 This ensures that the input value to \( G \) in \( R^{(4)} \) will be zero. XORing it with \( S^{-1}(x^{(3)}_L \oplus k^{(3)}_6) \), we can obtain the output value of \( G \) in \( R^{(4)} \). Next, we also observe that the input to \( R^{(4)} \) can be computed using \( G(x^{(3)}_L, k^{(3)}_6) \oplus x^{(3)}_R \).

Using a similar technique in Step 4, we can select 16 pairs of \((x^{(3)}_L, x^{(3)}_R)\) to construct another filter with filtering capability of \( 2^{-54.99} \). As we only have \( 2^{10.01} \) key candidates left, the probability that a wrong key that passes the filter is \( 2^{-44.98} \). After completing the above steps, we should be able to recover a total of 90 bits of key information:

- 60 bits of \( K^{(1)} \)
- 5 bits of \( k^{(5)}_6 \)
- 5 bits of \( k^{(4)}_i \) \( \forall i \in \{1, ..., 5\} \)

6. The remaining 30 bits of key information can be exhaustively searched.

**Complexity analysis.** The MitM-based key recovery attack on 4+4 rounds of SCARF will recover 60 bits of \( K^{(1)} \) and 60 bits of \( K^{(2)} \). The data complexity is pretty straightforward; we need the full codebook, i.e. \( 2^{10} \) pairs of \((P_1, P_2)\). As for the time complexity, we have to first discuss about the relative cost of checking the consistency of a linear system compared to a SCARF round function. According to [7, Proposition 1], Gaussian Elimination over \( \mathbb{F}_2 \) requires

\[
m \cdot n^2 + n^2 - n
\]

operations. For \( m = 16 \) and \( n = 5 \), the total number of bitwise operations required is 420. On the other hand, in the SCARF round function, the current data of left branch going through \( G \) function requires 6 AND and 5 XOR operations acting on 5-bit data, and then is XORed with the right branch. Thus to calculate

\[10\] We can always find such a pair as we have the entire codebook of \((x^{(3)}_L, x^{(3)}_R)\).
the newer right branch, it requires $5 \times 6 \times 2 = 60$ bit operations. While for calculating the newer left branch, the current data of left branch first XORs with a sub-tweakey and then goes through $S$. As $S$ contains 4 OR, 2 AND and 1 XOR bit operations (as the NOT gates are much cheaper in terms of latency, we do not consider them here), finally, one SCARF round needs in total $60 + 5 \times (1 + 4 + 2 + 1) = 100$ bit operations. Therefore, we approximate the cost of checking the consistency of a linear system with 16 equations and 5 variables to be 4.2 SCARF encryption rounds.

The time complexity is composed of the following four parts,

1. Generating the full codebook will require $2^{10} \times 2$ 4-round SCARF encryptions.
2. For the first filter, we have to check for each of the $2^{60}$ keys of $K^{(1)}$. First, we have to obtain the inputs and outputs of $G$ in $R^{(3)}$. This costs two sboxes and three 5-bit XOR operations which is less than 2 SCARF encryption rounds. Next, there are actually 5 sub-filters (corresponding to the 5 different linear system). We will only check if a key passes the second sub-filter if it passes the first one. Thus the cost is about

$$(2 + 4.2) \times (2^{60} + 2^{60-10.99} + 2^{60-10.99 \times 2} + 2^{60-10.99 \times 3} + 2^{60-10.99 \times 4}) \approx 2^{62.63}$$

SCARF rounds or $2^{60.63}$ 4-round SCARF encryption.
3. At this point, we are expecting about $2^{5.01}$ keys that pass the filter. For filter 2, we will need to guess $k_6^{(3)}$, and check for each of the remaining keys:

$$4.2 \times (2^{5.01} \times 2^5)$$

4. The time complexity of the final exhaustive search of the remaining 30 key bits is $2^{30}$.

In total, we can see that the majority of the time is spent on the first filter. We approximate the time complexity to be about the same as performing $2^{60.63}$ 4-round SCARF encryptions.

**Experimental verification.** We have conducted experiments on our key-recovery attacks to validate the effectiveness of our attack; the source code is available at https://gitfront.io/r/user-9335734/nRTZjZpB2EPd/SCARF. The main steps in the attack, that is, steps 1 to 4 of the key recovery phase, have been experimentally verified. To ensure that the experiments are computationally feasible, we only attempted to recover the eight least significant bits of $K^{(1)}$, and the remaining 52 bits are assumed to be known.

**Experiment Analysis.** According to the discussion in step 4 of the key recovery phase, the theoretical probability of $\Pr[C] \leq 2^{-10.998}$ when $m = 16$. Therefore, on average, a wrong key will survive in a single trial with a probability of at most $(2^8 - 1) \times 2^{-10.998} \approx 0.125$. Thus, the theoretical probability

---

11 In the case where we are attacking from round 5 to 8, then $2^{65.63}$ 4-round SCARF encryptions will be required.
of successfully recovering the correct key is at least 0.875. In our experiment, using 10,000 trials, we succeeded in recovering the keys with a probability of about 89.3%. The results show our experimental results basically agree with the theoretical analysis.

3.4 Key-recovery Attack under Security Requirement 1

Our attack actually works under the Security Requirement 1 with some relatively cheap online precomputations. Recall that our attack requires the whole codebook for $\tilde{E}_{T_1, T_2}$, i.e., given all $P \in \mathbb{F}_2^{10}$, we need to know $C = \tilde{E}_{T_1, T_2}(P)$. According to Security Requirement 1, we can know $E_{T_1}(P_1) = E_{T_2}(P_2)$ when it occurs. Thus, we can construct the codebook as follows,

1. Initialize a table $T$;
2. Choose any $T_1, T_2$;
3. For each $(P_1, P_2) \in \mathbb{F}_2^{10} \times \mathbb{F}_2^{10}$, ask the oracle to see if $E_{T_1}(P_1) = E_{T_2}(P_2)$ occurs;
4. If a collision occurs, set $T[P_1] = P_2$.

The table $T$ is then the codebook. The precomputation costs $2^{21}$ SCARF encryptions, and thus it is negligible compared to our main attack.

4 Conclusion

In this paper, we give a key-recovery attack on 4+4 rounds of SCARF under the single tweak setting. Our attack shows that an MitM attack can recover 120 key bits with $2^{60.63}$ SCARF encryptions. Under the single-tweak setting, our attack works for both Security Requirements 1 and 2 that are given by the designers. We believe our attack is crucial for a deeper understanding for designing ciphers that are unconventional like SCARF.

Acknowledgements. Firstly, we would like to express our sincere thanks to the organizers of ASK 2023, the 10th Asian Workshop on Symmetric Key Cryptography, where this collaboration was initiated. We also appreciate the valuable contributions of Mohamed Rachidi and Zhang Jianing, who actively participated in discussions during ASK 2023 but chose not to continue thereafter. Finally, we extend our gratitude to Yosuke Todo, whose insights helped us realize that we could extend our attack to work under Security Requirement 1 (see Section 3.4).

References


